

# Konfigurierbare Systemsoftware (KSS)

## VL 6 – Variability Management in the Large: The VAMOS/CADOS Approach

**Daniel Lohmann**

Lehrstuhl für Informatik 4  
Verteilte Systeme und Betriebssysteme

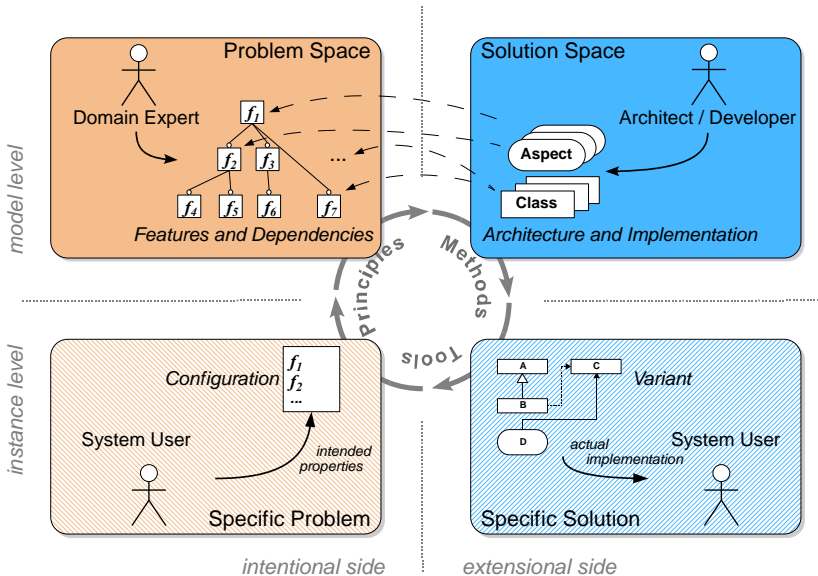
Friedrich-Alexander-Universität  
Erlangen-Nürnberg

SS 16 – 2016-05-30

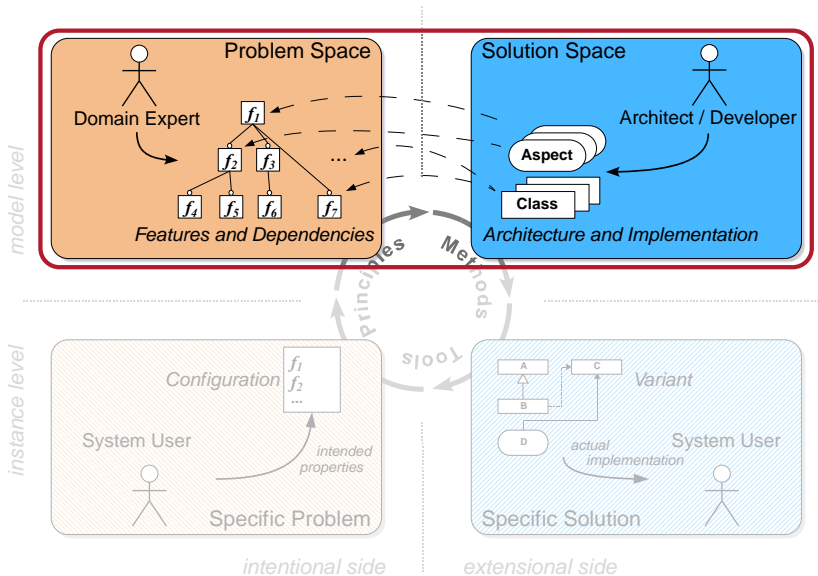
[http://www4.informatik.uni-erlangen.de/Lehre/SS16/V\\_KSS](http://www4.informatik.uni-erlangen.de/Lehre/SS16/V_KSS)



# About this Lecture



# About this Lecture

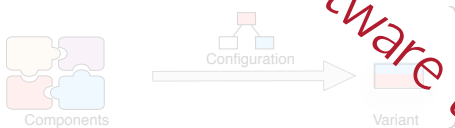


## Decompositional Approaches



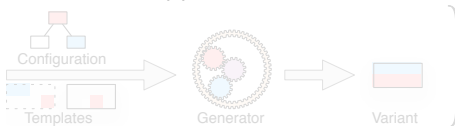
- Text-based filtering (untyped)
- Preprocessors

## Compositional Approaches



- Language-based composition mechanisms (typed)
- OOP, AOP, Templates

## Generative Approaches



- Metamodel-based generation of components (typed)
- MDD, C++ TMP, generators

Real-world software uses them all!!



# Agenda

---

- 6.1 Motivation
- 6.2 Variability in Linux
- 6.3 Configuration Consistency
- 6.4 Configuration Coverage
- 6.5 Automatic Tailoring
- 6.6 Summary
- 6.7 References



# 33 optional, independent features



one individual variant  
for each human being

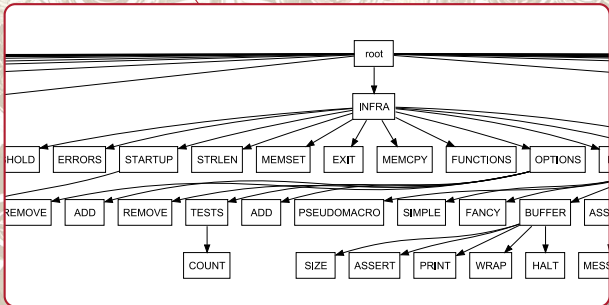
320 optional, independent  
features

more variants than  
atoms in the universe!

# Typical Configurable Operating Systems...



1,250 features





# Typical Configurable Operating Systems...

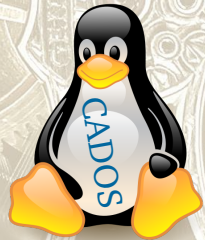


1,250 features

**Challenges:** → **VAMOS/CADOS\***

- How to maintain this?
- How to test this?
- Why so many features anyway?

\* Variability Management in Operating Systems  
Configurability-Aware Development of Operating Systems



12,000 features

6.1 Motivation

6.2 Variability in Linux

Variability Implementation in Linux

Challenges

6.3 Configuration Consistency

6.4 Configuration Coverage

6.5 Automatic Tailoring

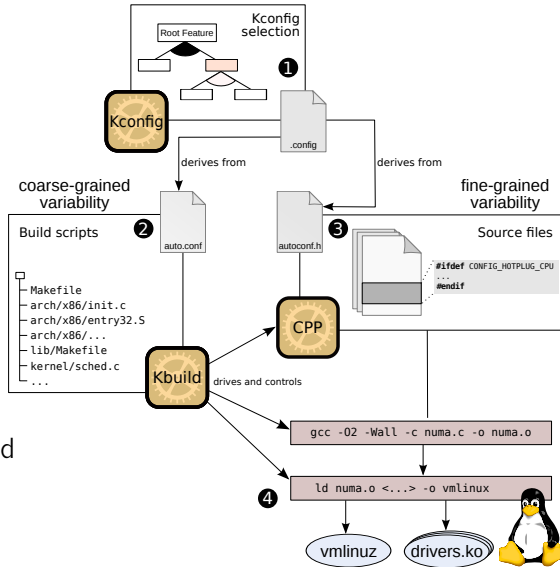
6.6 Summary

6.7 References



# The Linux Configuration and Generation Process

- 1 Configuration with an KCONFIG frontend
- 2 Compilation of a subset of files
- 3 Selection of a subset of CPP Blocks
- 4 Linking of the kernel and loadable kernel modules



$l_0$ : Feature Modeling 12,000 features

$l_1$ : Coarse-grained: KBUILD 31,000 source files

$l_2$ : Fine-grained: CPP 89,000 `#ifdef` blocks

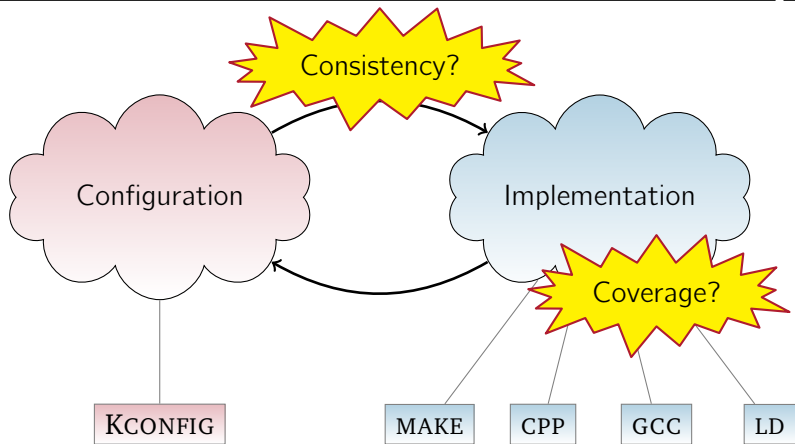
$l_3$ : Language-level: GCC  $\rightarrow$  `if(CONFIG_SMP) ...`

$l_4$ : Link time: LD  $\rightarrow$  branches in linker scripts

$l_5$ : Run time: INSMOD, MODPROBE, ...



# Challenges with Implemented Variability



- Central declaration of configurability: KCONFIG
- Distributed implementation of configurability: MAKE, CPP, GCC, LD



6.1 Motivation

6.2 Variability in Linux

6.3 Configuration Consistency

Problem Analysis

Solution Approach

Results

6.4 Configuration Coverage

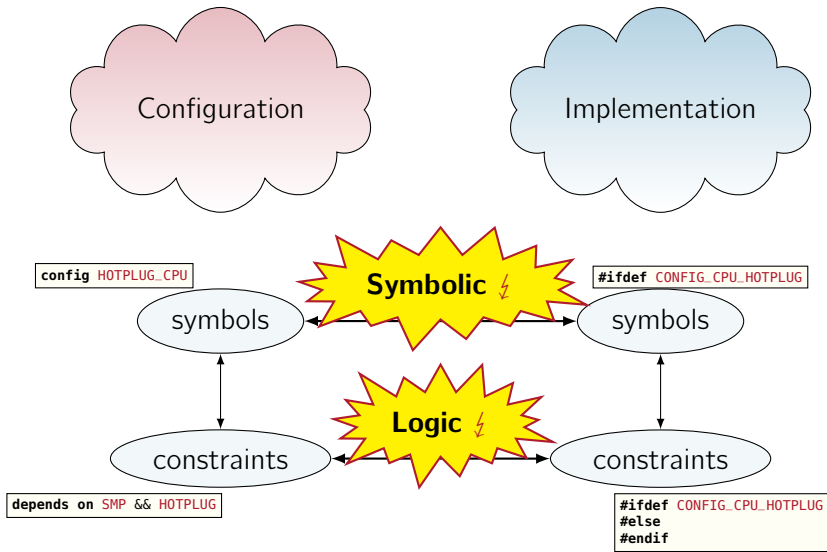
6.5 Automatic Tailoring

6.6 Summary

6.7 References



# Problem Analysis: Configuration Consistency



```
config HOTPLUG_CPU
    bool "Support for hot-pluggable CPUs"
    depends on SMP && HOTPLUG
    ---help---
```

```
static int
hotplug_cfd(struct notifier_block *nfb, unsigned long action, void *hcpu)
{
    // [...]
    switch (action) {
        case CPU_UP_PREPARE:
        case CPU_UP_PREPARE_FROZEN:
        // [...]
#ifdef CONFIG_CPU_HOTPLUG
        case CPU_UP_CANCELED:
        case CPU_UP_CANCELED_FROZEN:

        case CPU_DEAD:
        case CPU_DEAD_FROZEN:
            free_cpumask_var(cfd->cpumask);
            break;
#endif
    };
    return NOTIFY_OK;
}
```

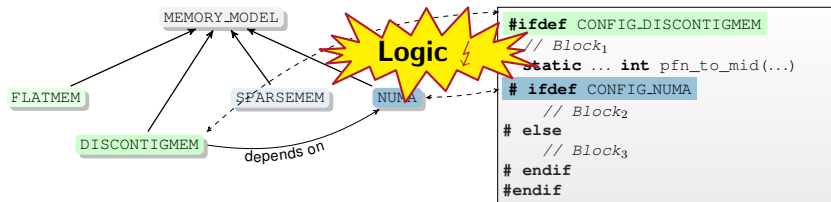
**Symbolic** ⚡

**Result:**

Fix for a  
critical bug







- Feature DISCONTIGMEM **implies** feature NUMA
  - Inner blocks are not actually configuration-dependent
    - *Block*<sub>2</sub> is **always** selected       $\mapsto$  **undead**
    - *Block*<sub>3</sub> is **never** selected       $\mapsto$  **dead**
- } **configurability defects**

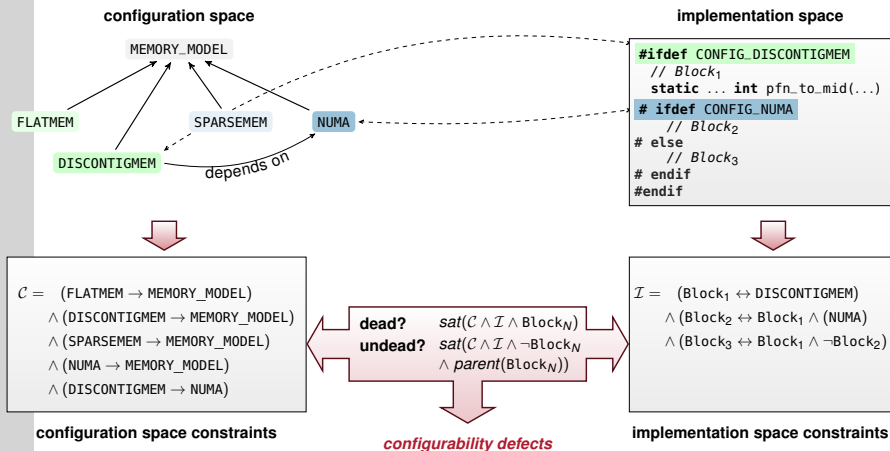
~ Linux contains **superfluous** #ifdef Blocks!

**Result:**  
Code cleanup



# Solution Approach: Consistency Validation

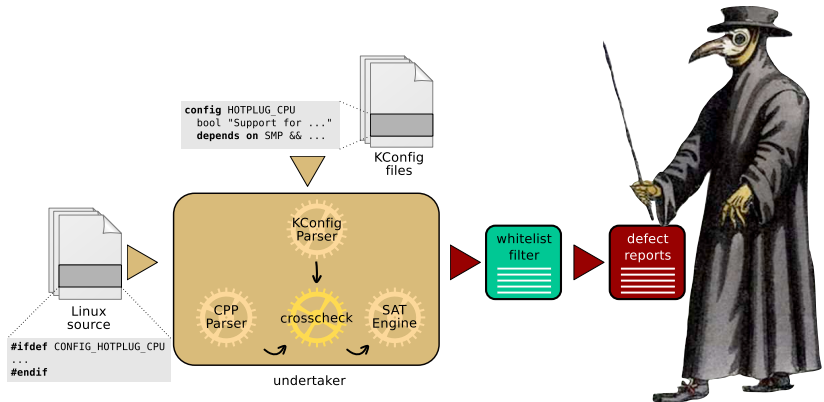
Problem and solution space are analyzed for configuration points:



⇒ and transformed into **propositional formulas**



**Job:** Find (and eventually bury) **dead #ifdef-code!**

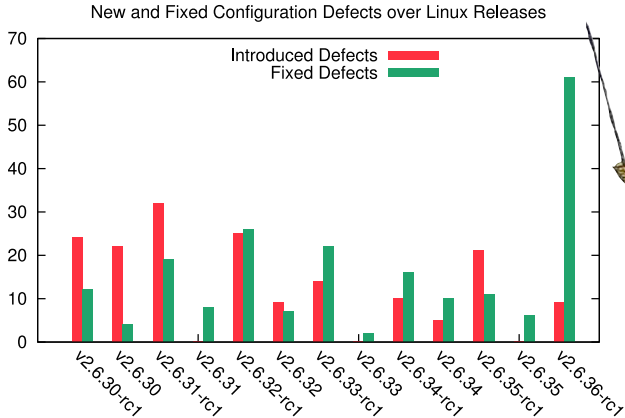


**Job:** Find (and eventually bury) **dead #ifdef-code!**

- We have found **1776** configurability defects in Linux v2.6.35
- Submitted **123** patches for **364** defects
- **20** are confirmed **new bugs** (affecting binary code)
- Cleaned up **5129** lines of cruft code



**Job:** Find (and eventually bury) **dead #ifdef-code!**



How good is this, *really*?



6.1 Motivation

6.2 Variability in Linux

6.3 Configuration Consistency

6.4 Configuration Coverage

Where Have All the Features Gone?

Results

Extracting Variability from KBUILD

Improvements

Implementation Space Coverage

6.5 Automatic Tailoring

6.6 Summary

6.7 References



# Common Beliefs About Variability in Linux

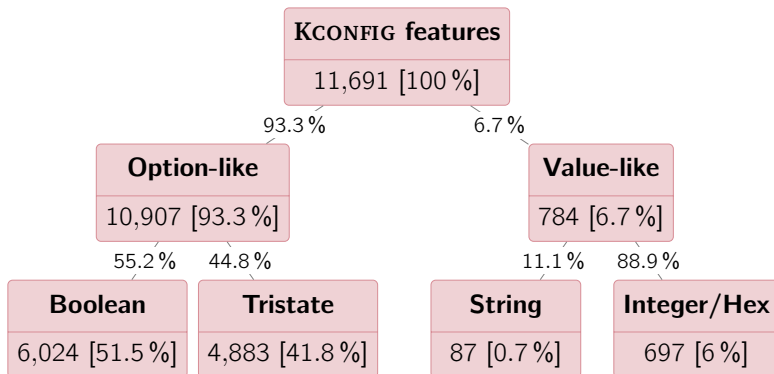
---

- ❶ Most variability is expressed by boolean (or tristate) switches.
- ❷ arch-x86 is the largest and allyesconfig selects most features.
- ❸ Variability is mostly implemented with the CPP.
- ❹ The Linux *kernel* is highly configurable.



# Linux v3.1: Feature Distribution by Type

- 1 Most variability is expressed by boolean (or tristate) switches



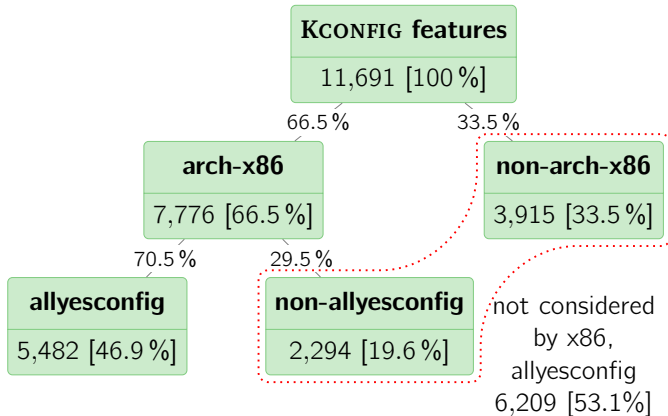
⇒ Almost all features in Linux are **option-like**





# Linux v3.1: Coverage of arch-x86 / allyesconfig

- ② arch-x86 is the largest and allyesconfig selects most features

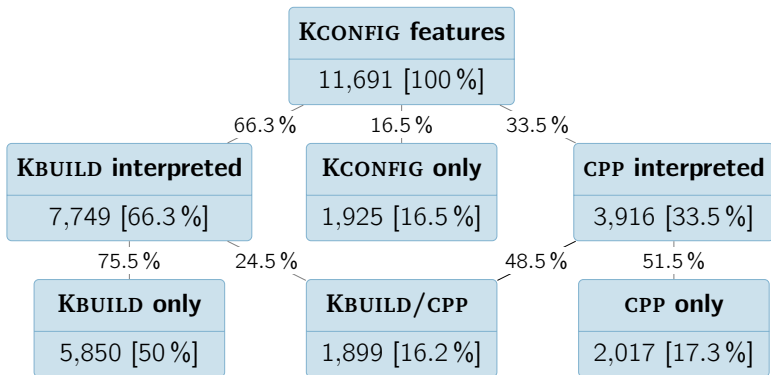


⇒ arch-x86/allyesconfig is **not nearly** a full configuration



# Linux v3.1: Distribution by Granularity

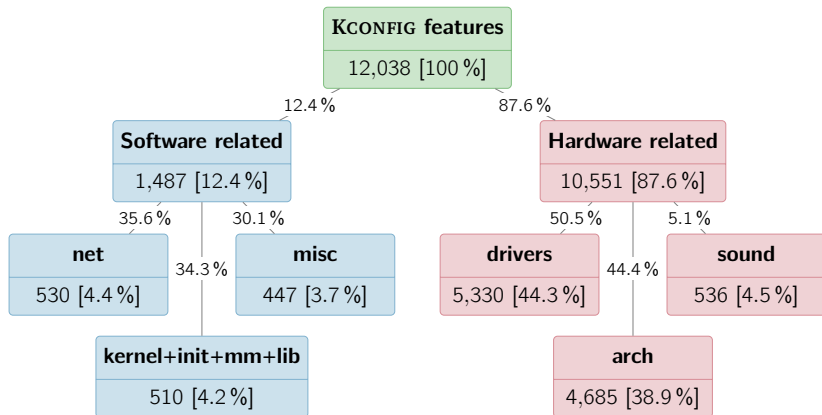
## ③ Variability is mostly implemented with the CPP



⇒ KBUILD implements **more than two thirds** of all variation points



## ④ The Linux *kernel* is highly configurable

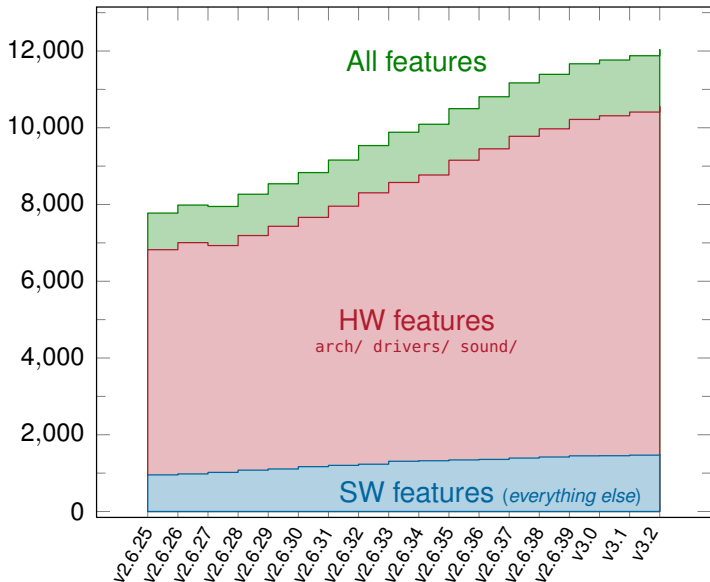


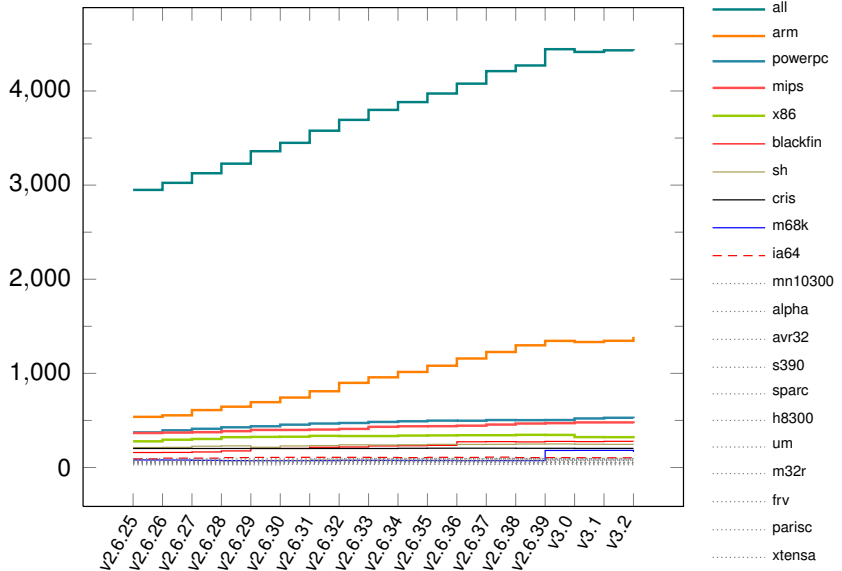
⇒ Software features account for  
**only twelve percent** of all variation points



# Linux Feature Growth over Time

(#Features, 2007–2012)





# Results: Where Have all the Features Gone?

- ① Most variability is expressed by boolean (or tristate) switches ✓
  - more than 93 percent of all features are option-like
    - ↪ it is acceptable for tools to ignore value-type features
- ② arch-x86 is the largest and allyesconfig selects most features ✗
  - more than 53 percent are not covered by this configuration
    - ↪ other parts of Linux are probably less tested and error-prone!
- ③ Variability is mostly implemented with the CPP ✗
  - more than 66 percent of all features are handled by the build system, only 17 percent are handled by CPP only
    - ↪ variability extraction from KBUILD is necessary
- ④ The Linux *kernel* is highly configurable ✗
  - only 12 percent of all features configure software only
  - variability is mostly induced by advances in hardware
    - ↪ complexity will increase further



## Challenges: Variability Extraction from the Build System

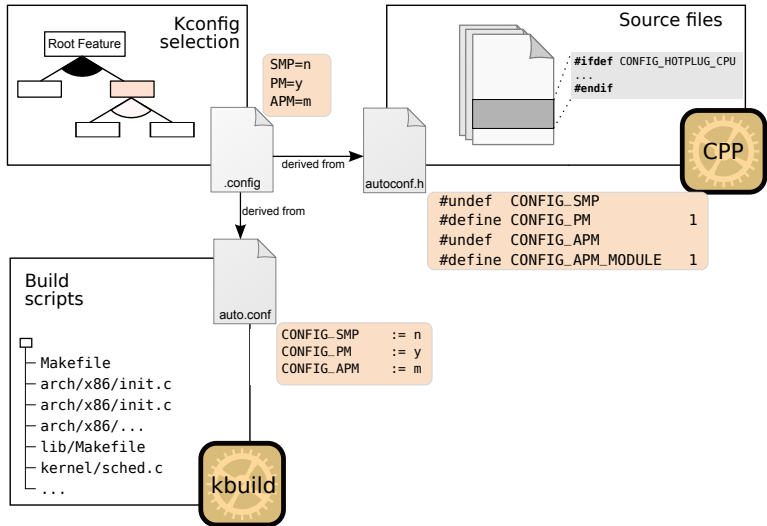
- Variability extraction  $\mapsto$  which file is selected by which feature?
- Usual approach for variability extraction [7, 11] (KCONFIG, CPP, ...):



- Parsing **does not work** well for MAKE-languages
  - declarative and Turing-complete languages
  - special features, like `shell`, `foreach`, `eval`, `addprefix`, ...
- Linux's KBUILD is built on top of (GNU) MAKE
  - nevertheless, researchers have tried parsing to extract variability
    - KBUILDMINER by Berger, She, Czarnecki, and Wasowski [1]
    - Nadi parser by Nadi and Holt [5]
  - resulting tools are too **brittle at best**
    - work for a (few) Linux version(s) only
    - each usage of a special feature requires manual tailoring



# Linux Build Process Revisited

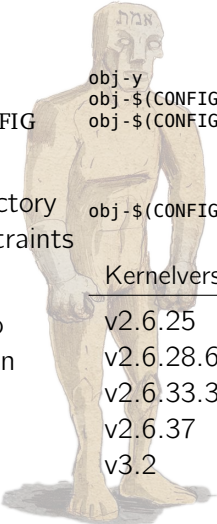




Basic idea: **Systematic probing and inferring of implications**

**SPLC '12:** Dietrich, et al. [2]

- *Dancing Makefiles*
  - Identification of KCONFIG references
  - Recursion into subdirectory while considering constraints
  - Robust with respect to architecture and version
- ⇒ **no adaptations** on or for KBUILD!

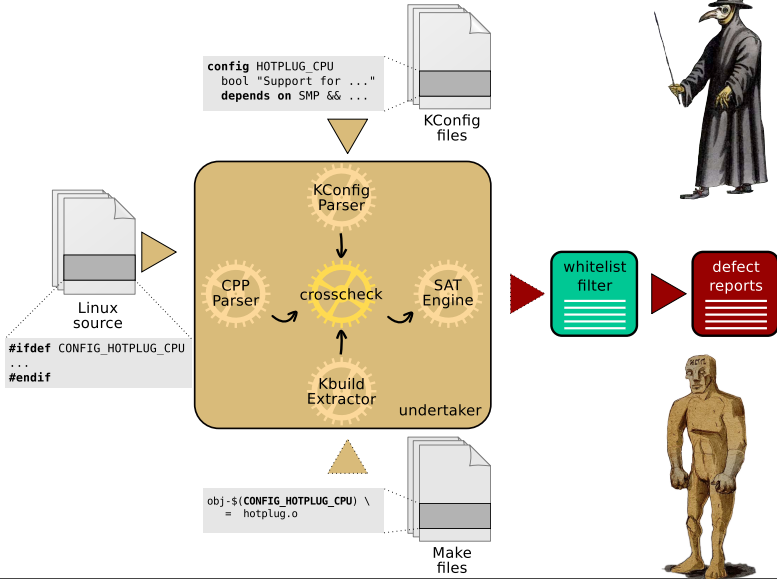


```
obj-y += fork.o
obj-$(CONFIG_SMP) += spinlock.o
obj-$(CONFIG_APM) += apm.o
```

```
obj-$(CONFIG_PM) += power/
```

Kernelversion	found inferences	
v2.6.25	6,274	(93.7%)
v2.6.28.6	7,032	(93.6%)
v2.6.33.3	9,079	(94.9%)
v2.6.37	10,145	(95.1%)
v3.2	11,050	(95.4%)





## Configuration defects in Linux v3.2:

### *Without KBUILD constraints*

Code defects	1835
Referential defects	415
Logical defects	83
Sum:	<b><math>\Sigma</math> 2333</b>

### *With KBUILD constraints*

Code defects	1835
Referential defects	439
Logical defects	299
Sum:	<b><math>\Sigma</math> 2573</b>



**Result: +10%**



## Issue: Decompositional Implementation of Variability

```
#ifdef CONFIG_NUMA
    Block1
#else
    Block2
#endif
```

Developer has to derive at least two configurations to ensure that the every line of code **even compiles!**

Make sure that the submitted code...

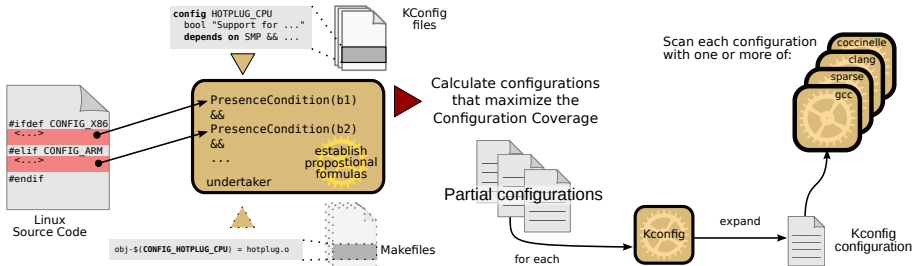
*“ 8. has been carefully reviewed with respect to relevant KCONFIG combinations. **This is very hard to get right with testing – brain-power pays off here.** ”*

Linux kernel patch submission checklist (Documentation/SubmitChecklist)



# The VAMPYR Driver for Static Checkers

- **Goal:** Maximize configuration coverage of *existing* tools
    - Every configuration-conditional part should be covered at least once
    - *Statement coverage*
- ⇒ Create a **set of configurations** and scan each individually



# The VAMPYR Driver for Static Checkers

**Goal:** Maximize configuration coverage of *existing* tools

- Even
- State

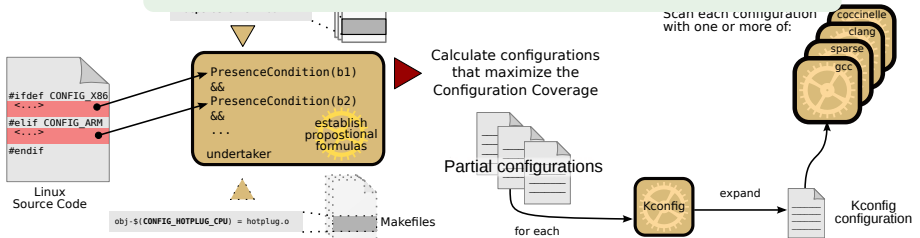
⇒ Create

```
$ git am bugfix.diff # Apply patch
```

```
$ vampyr -C gcc --commit HEAD # Examine
```

- Cover *each* conditional block on arch-arm:

```
$ vampyr -C gcc -b arm_worklist # nightly check
```



Software Project	allyesconf $CC_N$	VAMPYR $CC_N$	Overhead: increase of GCC Invocations	GCC #warnings VAMPYR (allyesconfig)	GCC #errors VAMPYR (allyesconfig)	$\Sigma$ Issues	#ifdef blocks per reported issue (bpi)	Result: increase of GCC messages
<b>Linux/x86</b>	78.6%	88.4%	21.5%	201 (176)	1 (0)	202	110	26 (+15%)
hardware	76.8%	86.5%	21.0%	180 (155)	1 (0)	181	82	26 (+17%)
software	82.7%	92.4%	22.7%	21 (21)	0 (0)	21	351	0 (+0%)
<b>Linux/arm</b>	59.9%	84.4%	22.7%	417 (294)	92 (15)	508	46	199 (+64%)
hardware	51.2%	80.1%	23.7%	380 (262)	92 (15)	471	34	194 (+70%)
software	83.6%	96.3%	19.5%	37 (32)	0 (0)	37	192	5 (+16%)
<b>Linux/mips</b>	54.5%	90.9%	22.0%	220 (157)	29 (1)	249	85	91 (+58%)
hardware	42.1%	88.2%	21.5%	174 (121)	17 (1)	191	72	69 (+57%)
software	79.8%	96.3%	23.2%	46 (36)	12 (0)	58	128	22 (+61%)
<b>L4/Fiasco</b>	99.1%	99.8%	see text	20 (5)	1 (0)	21	see text	16 (+320%)
<b>Busybox</b>	74.2%	97.3%	60.3%	44 (35)	0 (0)	44	72	9 (+26%)

### Example: arch-arm

- Increased CC compared to allyesconfig from **60% to 84%**
- **199 (+64%)** additional issues reported by GCC
- **91** reported issues have to be considered as **serious bugs**
- **7** patches submitted – all got immediately accepted

Just by letting **the compiler** see *all* the code!



6.1 Motivation

6.2 Variability in Linux

6.3 Configuration Consistency

6.4 Configuration Coverage

6.5 Automatic Tailoring

Idea

Results

6.6 Summary

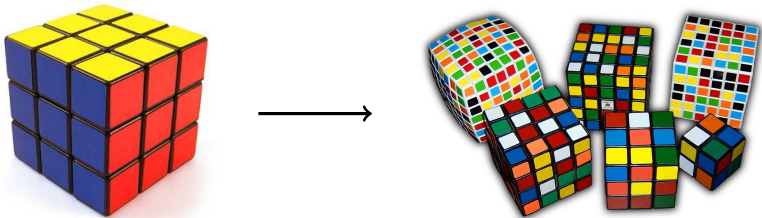
6.7 References





# Idea: Automated Tailoring of Linux

- Distribution kernels today come with a **maximum** configuration
- As side-effect, this maximizes the **attack** surface!
- Each use-case needs its specific, ideal configuration



→ Automatically derive an **ideal** configuration for a given use case.



# Automatic Tailoring: Approach



## Main idea: “measure” needed features

- Start with standard distribution kernel
- Run use-case-specific test load ➡ “observe” needed functionality
- Derive configuration for tailored kernel



# Automatic Tailoring: Approach

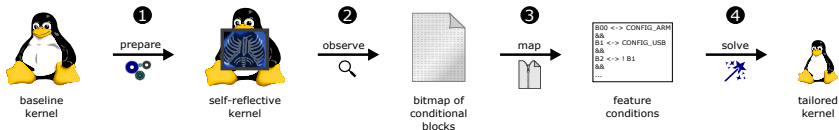
1

## Prepare feature tracing

- enable ftrace, or
- patch source with flipper



specific scenario



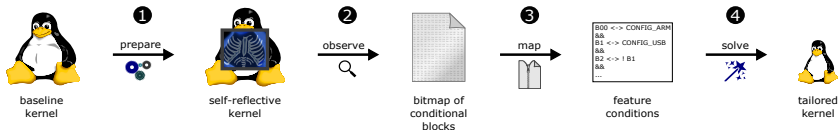
# Automatic Tailoring: Approach

- 1 Prepare feature tracing
  - enable ftrace, or
  - patch source with flipper
- 2 Run test load, observe
  - trace invoked kernel code
  - address  $\mapsto$  `#ifdef` block



specific scenario

test load



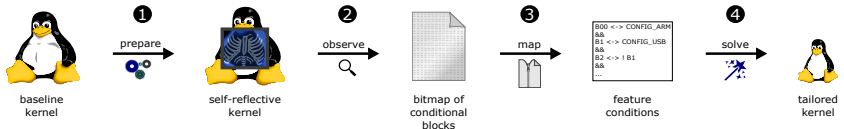
# Automatic Tailoring: Approach

- 1 Prepare feature tracing
  - enable ftrace, or
  - patch source with flipper
- 2 Run test load, observe
  - trace invoked kernel code
  - address  $\mapsto$  `#ifdef` block
- 3 Map to partial config
  - blocks  $\mapsto$  dependend blocks
  - blocks  $\mapsto$  features



specific scenario

test load



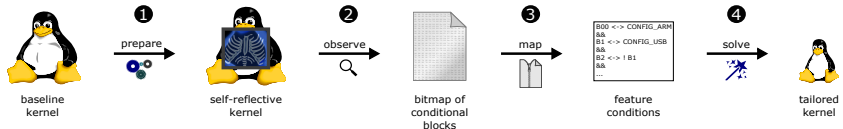
# Automatic Tailoring: Approach

- 1 Prepare feature tracing
  - enable ftrace, or
  - patch source with flipper
- 2 Run test load, observe
  - trace invoked kernel code
  - address  $\mapsto$  `#ifdef` block
- 3 Map to partial config
  - blocks  $\mapsto$  dependend blocks
  - blocks  $\mapsto$  features
- 4 Expand to full config
  - apply white/black list
  - resolve constraints

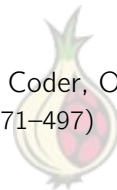
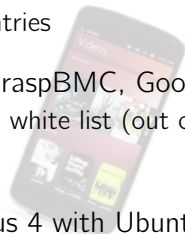


specific scenario

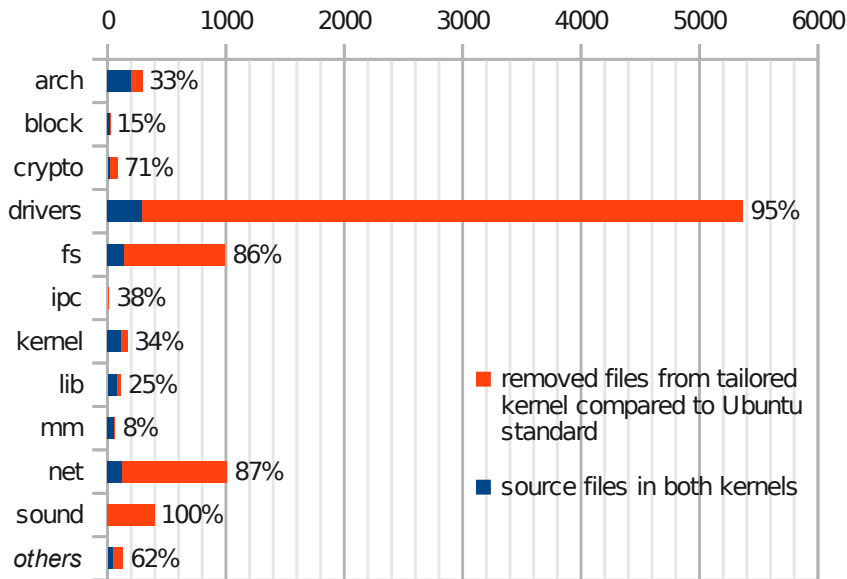
test load



- x86-based server/workstation systems (LAMP, Desktop with NFS)
  - 90% fewer features, 9 entries on white list (out of 495–555)
  - 90% less executable code
  - 10% fewer functions with CVE entries
- ARM-based low-cost appliances (raspBMC, Google Coder, Onion $\pi$ )
  - 70% fewer features, 14 entries on white list (out of 471–497)
  - 75% less executable code
- ARM-based high-end ASIC (Nexus 4 with Ubuntu Phone)
  - 30% fewer features, 14 entries on white list (out of 850)
  - 25% less executable code



# Evaluation: Reduction for LAMP





**HotDep '12:** Tartler, Kurmus, Ruprecht, Heinloth, Rothberg et al. [9]

- TCB is **significantly** smaller
- Easy to use: process is fully automated
- If necessary, the tailoring can be guided with whitelists and blacklists
- Going further: Dynamic ASR [4]
  - Even if present: Who is allowed to call what  $\rightsquigarrow$  CFG analysis
  - At runtime: Block illegal invocations.



- Real-world system software offers **thousands of features**
  - eCos: 1,250 features
  - Linux: 12,000 features } **mostly induced by hardware!**
  - central declaration (ecosConfig, KCONFIG)
  - distributed, multi-paradigm implementation (MAKE, CPP, GCC, ...)
- This imposes great challenges for management and maintenance
  - how to ensure configurability consistency?
  - how to ensure configuration coverage?
  - how to keep pace with the constant feature increase?
- A strong call for adequate tool support ↪ **VAMOS/CADOS**
  - already found **thousands** and fixed **hundreds** of defects and bugs
  - more to come!



- [1] Thorsten Berger, Steven She, Krzysztof Czarnecki, and Andrzej Wasowski. *Feature-to-Code Mapping in Two Large Product Lines*. Tech. rep. University of Leipzig (Germany), University of Waterloo (Canada), IT University of Copenhagen (Denmark), 2010.
- [2] Christian Dietrich, Reinhard Tartler, Wolfgang Schröder-Preikschat, and Daniel Lohmann. "A Robust Approach for Variability Extraction from the Linux Build System". In: *Proceedings of the 16th Software Product Line Conference (SPLC '12)*. (Salvador, Brazil, Sept. 2–7, 2012). Ed. by Eduardo Santana de Almeida, Christa Schwanninger, and David Benavides. New York, NY, USA: ACM Press, 2012, pp. 21–30. ISBN: 978-1-4503-1094-9. DOI: 10.1145/2362536.2362544.
- [3] Christian Dietrich, Reinhard Tartler, Wolfgang Schröder-Preikschat, and Daniel Lohmann. "Understanding Linux Feature Distribution". In: *Proceedings of the 2nd AOSD Workshop on Modularity in Systems Software (AOSD-MISS '12)*. (Potsdam, Germany, Mar. 27, 2012). Ed. by Christoph Borchert, Michael Haupt, and Daniel Lohmann. New York, NY, USA: ACM Press, 2012. ISBN: 978-1-4503-1217-2. DOI: 10.1145/2162024.2162030.



- [4] Anil Kurmus, Reinhard Tartler, Daniela Dorneanu, Bernhard Heinloth, Valentin Rothberg, Andreas Ruprecht, Wolfgang Schröder-Preikschat, Daniel Lohmann, and Rüdiger Kapitza. "Attack Surface Metrics and Automated Compile-Time OS Kernel Tailoring". In: *Proceedings of the 20th Network and Distributed Systems Security Symposium*. (San Diego, CA, USA, Feb. 24–27, 2013). The Internet Society, 2013. URL: [http://www.internetsociety.org/sites/default/files/03\\_2\\_0.pdf](http://www.internetsociety.org/sites/default/files/03_2_0.pdf).
- [5] Sarah Nadi and Richard C. Holt. "Mining Kbuild to Detect Variability Anomalies in Linux". In: *Proceedings of the 16th European Conference on Software Maintenance and Reengineering (CSMR '12)*. (Szeged, Hungary, Mar. 27–30, 2012). Ed. by Tom Mens, Yiannis Kanellopoulos, and Andreas Winter. Washington, DC, USA: IEEE Computer Society Press, 2012. ISBN: 978-1-4673-0984-4. DOI: 10.1109/CSMR.2012.21.
- [6] Andreas Ruprecht, Bernhard Heinloth, and Daniel Lohmann. "Automatic Feature Selection in Large-Scale System-Software Product Lines". In: *Proceedings of the 13th International Conference on Generative Programming and Component Engineering (GPCE '14)*. (Västerås, Sweden). Ed. by Matthew Flatt. New York, NY, USA: ACM Press, Sept. 2014, pp. 39–48. ISBN: 978-1-4503-3161-6. DOI: 10.1145/2658761.2658767.



- [7] Julio Sincero, Reinhard Tartler, Daniel Lohmann, and Wolfgang Schröder-Preikschat. "Efficient Extraction and Analysis of Preprocessor-Based Variability". In: *Proceedings of the 9th International Conference on Generative Programming and Component Engineering (GPCE '10)*. (Eindhoven, The Netherlands). Ed. by Eelco Visser and Jaakko Järvi. New York, NY, USA: ACM Press, 2010, pp. 33–42. ISBN: 978-1-4503-0154-1. DOI: 10.1145/1868294.1868300.
  
- [8] Reinhard Tartler, Christian Dietrich, Julio Sincero, Wolfgang Schröder-Preikschat, and Daniel Lohmann. "Static Analysis of Variability in System Software: The 90,000 #ifdefs Issue". In: *Proceedings of the 2014 USENIX Annual Technical Conference*. (Philadelphia, PA, USA). Berkeley, CA, USA: USENIX Association, June 2014, pp. 421–432. ISBN: 978-1-931971-10-2. URL: <https://www.usenix.org/system/files/conference/atc14/atc14-paper-tartler.pdf>.



- [9] Reinhard Tartler, Anil Kurmus, Bernard Heinloth, Valentin Rothberg, Andreas Ruprecht, Daniela Doreanu, Rüdiger Kapitza, Wolfgang Schröder-Preikschat, and Daniel Lohmann. "Automatic OS Kernel TCB Reduction by Leveraging Compile-Time Configurability". In: *Proceedings of the 8th International Workshop on Hot Topics in System Dependability (HotDep '12)*. (Los Angeles, CA, USA). Berkeley, CA, USA: USENIX Association, 2012, pp. 1–6. URL: <https://www.usenix.org/system/files/conference/hotdep12/hotdep12-final11.pdf>.
- [10] Reinhard Tartler, Daniel Lohmann, Christian Dietrich, Christoph Egger, and Julio Sincero. "Configuration Coverage in the Analysis of Large-Scale System Software". In: *ACM SIGOPS Operating Systems Review* 45.3 (Jan. 2012), pp. 10–14. ISSN: 0163-5980. DOI: 10.1145/2094091.2094095.
- [11] Reinhard Tartler, Daniel Lohmann, Julio Sincero, and Wolfgang Schröder-Preikschat. "Feature Consistency in Compile-Time-Configurable System Software: Facing the Linux 10,000 Feature Problem". In: *Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2011 (EuroSys '11)*. (Salzburg, Austria). Ed. by Christoph M. Kirsch and Gernot Heiser. New York, NY, USA: ACM Press, Apr. 2011, pp. 47–60. ISBN: 978-1-4503-0634-8. DOI: 10.1145/1966445.1966451.

